Reversibility in Chemical Reactions*

Stefan Kuhn[0000−0002−5990−4157]1, Bogdan Aman[0000−0001−7649−8181]2,3, Gabriel Ciobanu[0000−0002−8166−9456]2,3, Anna Philippou[0000−0002−1665−9913]4, Kyriaki Psara[0000−0002−6554−7950]4, and Irek Ulidowski[0000−0002−3834−2036]5

[1] School of Computer Science and Informatics, De Montfort University, Leicester, UK
stefan.kuhn@dmu.ac.uk
[2] Romanian Academy, Institute of Computer Science, Iaşi, România
[3] A.I.Cuza University, Faculty of Computer Science, Iaşi, România
{bogdan.aman, gabriel}@info.uaic.ro
[4] Department of Computer Science, University of Cyprus, Nicosia, Cyprus
{annap,kpsara01}@cs.ucy.ac.cy
[5] School of Informatics, University of Leicester, Leicester, UK
irek.ulidowski@leicester.ac.uk

Abstract. In this chapter we give an overview of techniques for the modelling and reasoning about reversibility of systems, including out-of-causal-order reversibility, as it appears in chemical reactions. We consider the autoprotolysis of water reaction, and model it with the Calculus of Covalent Bonding, the Bonding Calculus, and Reversing Petri Nets. This exercise demonstrates that the formalisms, developed for expressing advanced forms of reversibility, are able to model autoprotolysis of water very accurately. Characteristics and expressiveness of the three formalisms are discussed and illustrated.

Keywords: Reversible Computation · Reaction Modelling · Calculus of Covalent Bonding · Bonding Calculus · Reversing Petri Nets

1 Introduction

Biological reactions, pathways, and reaction networks have been extensively studied in the literature using various techniques, including process calculi and Petri nets. Initial research was mainly focused on reaction rates by the modelling and simulating networks of reactions, in order to analyse or even predict the common paths through the network. Reversibility was not considered explicitly. Later on reversibility started to be taken into account, since it plays a crucial rôle in many processes, typically by going back to a previous state in the system. Two common types of reversibility are backtracking and causally-consistent reversibility [8, 25, 19]. Backtracking executes exactly the inverse order of the forward execution, and causally-consistent reversibility allows undoing effects before causes, but not necessarily in the exact inverse order. Beyond backtracking and causally-consistent reversibility, there is a more general form of

* The authors acknowledge partial support of COST Action IC1405 on Reversible Computation - Extending Horizons of Computing.

reversibility, known as out-of-causal-order reversibility [28], which makes it possible to get to states which cannot be reached by forward reactions alone. Such sequences of forward and reverse reaction steps are important as they lead to new chemical structures and new reactions, which would not be possible without out-of-causal-order reversibility [28]. A typical example is a catalytic reaction: a catalyst C enables compounds A and B to combine, a combination that would not normally happen or be very unlikely without the presence of C. Initially, catalyst C binds with B resulting in a compound BC. Then A combines with BC creating ABC. Finally, with its job done, C breaks away from ABC, leaving A and B bonded. This sequence of reactions can be written as follows:

$$A + B + C \rightarrow A + BC \rightarrow ABC \rightarrow AB + C$$

This is a typical example of out-of-causal order reversibility since the bond between B and C is undone before its effect, namely the bond from A to B (which is not undone at all). The modelling of such reactions is the focus of this chapter. For further motivation, formal definitions and more illustrating examples of the various types of reversibility we refer the reader to [8, 25, 19, 28].

1.1 Contribution

This chapter presents and compares three formalisms, the Calculus of Covalent Bonding (CCB) [15, 16], the Bonding Calculus [1], and Reversing Petri Nets [23], that have been developed during COST Action IC1405. These models are variations of existing formalisms and set out to study reversible computation by allowing systems to reverse at any time leading to previously visited states or even new states without the need of additional forward actions. The contribution of this chapter is a comparative overview of the three formalisms, a discussion of their expressiveness, and a demonstration of their use on a common case study, namely the autoprotolysis of water reaction.

Our case study was selected to be non-trivial, of manageable size, and to allow us to exhibit the crucial features of the formalisms. It is a chemical reaction that involves small molecules, so it is different from biological reactions that involve proteins and other macromolecules. New modelling techniques may be needed in order to capture fully reversible behaviour of biological systems, however, in this chapter we concentrate on chemical reactions, a domain that offers interesting examples of out-of-causal-order reversibility.

The discussed formalisms enable us to model the intermediate steps of chemical reactions where some bonds are only "helping" to achieve the overall aim of the reaction: specifically, they are only formed to be broken before the end of the reactions. Thus, the allowed level of detail makes a more accurate depiction of the reversibility possible, and allows a more thorough understanding of the underlying reaction mechanisms compared to higher-level models.

1.2 Related Work

Process calculi, originally designed for the modelling of sequential and concurrent computation, have been applied to biochemical and biological systems. The main instances are the π-calculus [34], BioAmbients [33], the stochastic π-calculus [30], beta binders [31] and bioPEPA [6]. Another way to model biochemical reactions is with rule-based formalisms such as BIOCHAM [10], the κ-calculus [7], and the BioNetGen Language (BNGL) [9]. The formalisms κ and BNGL can be used to model interactions between proteins, while this is not possible in BIOCHAM. BNGL allows the use of molecule sites having the same name, which is not allowed in the κ-calculus.

Most of the formalisms mentioned above do not explicitly represent reversibility. If an action is the reverse of another action performed before, there is no explicit knowledge of that in the model. Reversibility was added explicitly to process calculi in RCCS [8], CCSK [25], and reversible π [17, 18]. CCSK and RCCS are based on the Calculus of Communicating Systems (CCS) [21]. They extend CCS by keeping track of past actions and enabling an undo of those. So a reverse action is the reverse execution of a forward action. These calculi support backtracking and causally-consistent reversibility. Out-of-causal-order reversibility was first addressed in CCSK extended with controller processes [28], and in the context of reversible event structures [27, 26, 37]. CCB [16] allows all types of reversibility in the context of chemical reactions and in other settings.

Petri nets (PNs) [35] are another formalism that has been widely used to model and reason about a wide range of applications featuring concurrency and distribution. They are a graphical language associated with a rich mathematical theory and supported by a variety of tools. Their use in systems biology dates back to [32, 12]. Since then, they have been employed for the modelling, analysis, and simulation of biochemical reactions in metabolic pathways, gene expression, signal transduction, and neural processes [4, 5, 2]. Indeed, PNs seem to be a natural framework for representing biochemical systems as they constitute a set of interdependent transitions/reactions which consume and produce resources, and are represented graphically in a similar fashion to the systems in question. Several specialised Petri net classes, such as qualitative, stochastic, continuous, or hybrid Petri nets and their coloured counterparts, have been used to describe different biochemical systems [22, 13, 29, 20, 38].

Even though classical PNs and their extensions have been extensively used to model biochemical systems, they cannot directly model reversibility. Specifically, when modelling reversible reactions in these formalisms it is required to employ mechanisms involving two distinct transitions, one for the forward and one for the reverse version of a reaction. This may result in expanded models and less natural and/or less accurate models of reversible behaviour. It is also in contrast to the notion of reversible computation, where the intention is not to return to a state via arbitrary execution but to reverse the effect of already executed transitions. For this reason, the formalism of reversing Petri nets [23] has been proposed to allow systems to reverse already executed transitions leading to previously visited states or even new ones without the need of additional forward actions.

Reversing Petri nets have also been extended with a mechanism for controlling transition reversal by associating transitions with conditions [24].

1.3 Paper organisation

In the next section, we introduce the autoprotolysis of water reaction, which will be modelled using our three formalisms. This is followed by a section introducing the formalisms, their syntax and, informally, their operational semantics. We also give three models of the autoprotolysis of water using the formalisms. In Section 4, we compare the formalisms and the models of our example reaction, and we also briefly discuss software support for the three formalisms. Finally, Section 5 concludes the paper.

2 Autoprotolysis of Water

We consider a chemical reaction that transfers a hydrogen atom between two water molecules. This reaction is known as the *autoprotolysis of water* and is shown in Figure 1. There, O indicates an oxygen atom and H a hydrogen atom. The lines indicate bonds. Positive and negative charges on atoms are shown by $\oplus$ and $\ominus$ respectively. The meaning of the curved arrows and the dots will be explained in the next paragraphs. The reaction is reversible and it takes place at a relatively low rate, making pure water slightly conductive. We have chosen this reaction as our example reaction, since it is non-trivial but manageable, and has some interesting aspects to be represented.

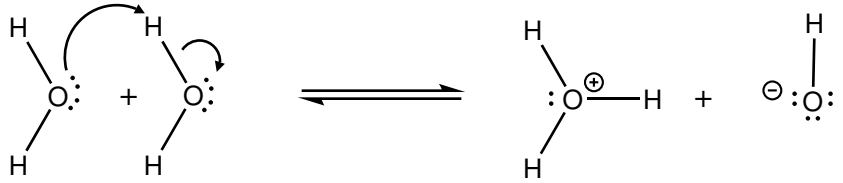

Fig. 1. Autoprotolysis of water.

To model the reaction we need to understand why it takes place and what causes it. The main reason is that the oxygen in the water molecule is *nucleophilic*, meaning it has the tendency to bond to another atomic nucleus, which would serve as an *electrophile*. This is because oxygen has a high electronegativity, therefore it attracts electrons and has an abundance of electrons around it. The electrons around the atomic nucleus are arranged on electron shells, where only those in the outer shell participate in bonding. Oxygen has four electrons in its outer shell, which are not involved in the initial bonding with hydrogen atoms. These electrons form two *lone pairs* of two electrons each, which can form new bonds (lone pairs are shown in Figure 1 by pairs of dots). All this makes oxygen nucleophilic: it tends to connect to other atomic nuclei by

forming bonds from its lone pairs. Since oxygen attracts electrons, the hydrogen atoms in water have a positive partial charge and oxygen has a negative partial charge.

The reaction starts when an oxygen in one water molecule is attracted by a hydrogen in another water molecule due to their opposite charges. This results in a *hydrogen bond*. This bond is formed out of the electrons of one of the lone pairs of the oxygen. The large curved arrow in Figure 1 indicates the movements of the electrons. Since a hydrogen atom cannot have more than one bond, the creation of a new bond is compensated by breaking the existing hydrogen-oxygen bond (indicated by the small curved arrow). When this happens, the two electrons, which formed the original hydrogen-oxygen bond, remain with the oxygen. Since a hydrogen contains one electron and one proton, it is only the proton that is transferred, so the process can be called a proton transfer as well as a hydrogen transfer. The forming of the new bond and the breaking of the old bond are *concerted*, meaning that they happen together without a stable intermediate configuration. As a result we have reached the state where one oxygen atom has three bonds to hydrogen atoms and is positively charged, represented on the right side of the reaction in Figure 1. This molecule is called *hydronium* and is written as H_3O^+. The other oxygen atom bonds to only one hydrogen and is negatively charged, having an electron in surplus. This molecule is called a *hydroxide* and is written as OH^-.

Note that the reaction is reversible: the oxygen that lost a hydrogen can pull back one of the hydrogens from the other molecule, the H_3O^+ molecule. This is the case since the negatively charged oxygen is a strong nucleophile and the hydrogens in the H_3O^+ molecule are all positively charged. Thus, any of the hydrogens can be removed, making both oxygens formally uncharged, and restoring the two water molecules. In Figure 1 the curved arrows are given for the reaction going from left to right. Since the reaction is reversible (indicated by the double arrow) there are corresponding electron movements when going from right to left. These are not given in line with usual conventions, but can be inferred.

In this simple reaction, the forward and the reverse step consist of two steps each. The breaking of the old and the forming of the new bond occur simultaneously. This means that there is no strict causality of actions, since none of them can be called the cause of the overall reaction. Furthermore, the reverse step can be done with a different atom to the one used during the forward step because each of the molecules are in a sense identical and in practice there does not exist a single "reverse" path corresponding to a forward one.

It should be noted that there are two types of bonding modelled here. Firstly, we have the initial bonds where two atoms contribute an electron each. Secondly, the *dative* or *coordinate bonds* are formed where both electrons come from one atom (an oxygen in this case). Both are *covalent bonds*, and once formed they cannot be distinguished. Specifically, in the oxygen with three bonds all bonds are the same and no distinction can be made. If one of the bonds is broken by a deprotonation (as in the autoprotolysis of water) the two electrons are left

behind and they form a lone pair. If the broken bond was not previously formed as a dative bond, the electrons changed their "rôle". This explains why any proton can be transferred in the reverse reaction and not just the one that was involved in the forward path.

3 Formalisms for Reversible Chemical Reactions

3.1 Calculus of Covalent Bonding

In this subsection we introduce the Calculus of Covalent Bonding (CCB) [16], concentrating on the new general prefixing operator $(s;b).P$ which, together with a generalised composition operator, produces pairs of *concerted* actions. Then we present a CCB model of the autoprotolysis of water.

Definition of CCB We recall the definition of CCB, presenting only the main ideas. More details can be found in [15, 16]. First, we introduce some preliminary notions and notations.

Let $\mathcal{A}$ be the set of (forward) action labels, ranged over by a, b, c, d, e, f. We partition $\mathcal{A}$ into the set of *strong actions*, written as $\mathcal{SA}$, and the set of *weak actions*, written as $\mathcal{WA}$. Reverse (or past) action labels are members of $\underline{\mathcal{A}}$, with typical members $\underline{a}, \underline{b}, \underline{c}, \underline{d}, \underline{e}, \underline{f}$, and represent undoing of actions. The set $\mathcal{P}(\mathcal{A} \cup \underline{\mathcal{A}})$ is ranged over by L.

Let $\mathcal{K}$ be an infinite set of *communication keys* (or *keys* for short) [25], ranged over by k, l, m, n. The Cartesian product $\mathcal{A} \times \mathcal{K}$, denoted by $\mathcal{AK}$, represents past actions, which are written as $a[k]$ for $a \in \mathcal{A}$ and $k \in \mathcal{K}$. Correspondingly, we have the set $\underline{\mathcal{A}}\mathcal{K}$ that represents undoing of past actions. We use α, β to identify actions which are either from $\mathcal{A}$ or $\mathcal{AK}$. It would be useful to consider sequences of actions or past actions, namely the elements of $(\mathcal{A} \cup \mathcal{AK})^*$, which are ranged over by s, s' and sequences of purely past actions, namely the elements of $\mathcal{AK}^*$, which are ranged over by t, t'. The empty sequence is denoted by ϵ. We use the notation "α,s" and "s,s'" to denote a concatenation of elements, which can be strings or single actions.

We shall also use two sets of auxiliary action labels, namely the set $(\mathcal{A}) = \{(a) \mid a \in \mathcal{A}\}$, and its product with the set of keys, namely $(\mathcal{A})\mathcal{K}$. These labels will be used in the auxiliary rules when defining the semantics of CCB. They denote the execution of a weak action, which makes it possible in the SOS rules to force breaking of a bond for those actions only.

The syntax of CCB is given below where P is a process term:

$$P ::= S \mid S \stackrel{def}{=} P \mid (s;b).P \mid P|Q \mid P\backslash L$$

The set of process identifiers (constants) $\mathcal{PI}$ contains typical elements S and T. Each process identifier S has a defining equation $S \stackrel{def}{=} P$ where P contains only forward actions (and no past actions). There is also a special identifier $\mathbf{0}$,

denoting the deadlocked process, which has no defining equation. For restrictions $L \subseteq \mathcal{A}$ holds.

We have a general prefixing operator $(s; b).P$, where s is a non-empty sequence of actions or past actions. This operator extends the prefixing operator in [28]. The action b is a weak action and it can be omitted, in which case the prefixing is written as $(s).P$ and is called the *simple prefix*. The simple prefix (which is still a sequence) is the prefixing operator in [28]. Exactly one of the actions in s in $(s).P$ may be a weak action from $\mathcal{WA}$. A weak action in s is only allowed for the simple prefix, in the $(s; b)$ operator b is the only allowed weak action. If s is a sequence that contains a single action, then the action is a strong action and the operator is the prefixing operator of CCS [21]. We omit trailing $\mathbf{0}$s so, for example, $(s).\mathbf{0}$ is written as (s). The new feature of the operator $(s; b).P$ is the execution of the weak action b, which can happen only after all the actions in s have taken place. Performing b then forces undoing one of the past actions in s (by the concert rule in Figure 4). If a $(s; b)$ operator is followed by another sequence of actions, where all actions in s have already taken place, then there is a non-deterministic choice of either doing b or progressing to the next sequence of actions (see act1 and act2).

$P \mid Q$ represents two systems P and Q which can perform actions or reverse actions on their own, or which can interact with each other according to a communication function γ. As in the calculus ACP [11], the communication function is a partial function $\gamma : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ which is commutative and associative. The function γ is used in the operational semantics to define when two processes can interact. Processes P and Q in $P \mid Q$ can also perform a pair of concerted actions, which is the new feature of our calculus. We also have the ACP-like restriction operator $\backslash L$, where L is a set of labels. It prevents actions from taking place and, due to the synchronisation algebra used, it also blocks communication. If $\gamma(a, b) = c$ then $a.P$ and $b.Q$ cannot communicate in $(a.P \mid b.Q) \setminus c$.

The set Proc of *process terms* is ranged over by P, Q and R. In the setting of CCB these terms are simply called *processes*. We define the semantics of our calculus using SOS rules (Figures 2–4) and rewrite rules (Figure 5).

We use some predicates and functions, which are formally defined in [16]. Informally, a process P is standard, written $\mathsf{std}(P)$, if it contains no past actions (hence no keys). A key n is fresh in Q, written $\mathsf{fsh}[n](Q)$, if Q contains no past action with the key n. Function k returns the keys in a sequence of actions, whereas keys returns the keys in a process, and fn gives the actions of a process which could be executed.

The forward and reverse SOS rules for CCB are given in Figures 2 and 3. Figure 4 contains the SOS rules that define the new concerted actions transitions. The rule concert defines when a pair of concerted actions takes place. This enables the linking of forming and breaking of bonds, and therefore a degree of control over the reversing of actions. The modelling in the next section will give examples of the application. Note that the concert rule uses *lookahead* [36]. Lookahead is a property of SOS rules, where a variable appears both on the right hand side and on the left hand side of a transition in the premises. for example P' and Q'

$$\textsf{act1}\ \frac{\textsf{std}(P) \quad \textsf{fsh}[k](s,s')}{(s,a,s';b).P \xrightarrow{a[k]} (s,a[k],s';b).P} \qquad \textsf{act2}\ \frac{P \xrightarrow{a[k]} P' \quad \textsf{fsh}[k](t)}{(t;b).P \xrightarrow{a[k]} (t;b).P'}$$

$$\textsf{par}\ \frac{P \xrightarrow{a[k]} P' \quad \textsf{fsh}[k](Q)}{P \mid Q \xrightarrow{a[k]} P' \mid Q} \qquad \textsf{com}\ \frac{P \xrightarrow{a[k]} P' \quad Q \xrightarrow{d[k]} Q'}{P \mid Q \xrightarrow{c[k]} P' \mid Q'}\ (*)$$

$$\textsf{res}\ \frac{P \xrightarrow{a[k]} P'}{P\backslash L \xrightarrow{a[k]} P'\backslash L}\ a \notin L \qquad \textsf{con}\ \frac{P \xrightarrow{a[k]} P'}{S \xrightarrow{a[k]} P'}\ S \stackrel{def}{=} P$$

Fig. 2. Forward SOS rules for CCB. The condition (*) is $\gamma(a,d) = c$, and $b \in \mathcal{WA}$. Recall that s is a sequence of actions and past actions and t is a sequence of purely past actions.

$$\textsf{rev act1}\ \frac{\textsf{std}(P)}{(s,a[k],s';b).P \xrightarrow{\underline{a}[k]} (s,a,s';b).P} \qquad \textsf{rev act2}\ \frac{P \xrightarrow{\underline{a}[k]} P'}{(t;b).P \xrightarrow{\underline{a}[k]} (t;b).P'}$$

$$\textsf{rev par}\ \frac{P \xrightarrow{\underline{a}[k]} P' \quad \textsf{fsh}[k](Q)}{P \mid Q \xrightarrow{\underline{a}[k]} P' \mid Q} \qquad \textsf{rev com}\ \frac{P \xrightarrow{\underline{a}[k]} P' \quad Q \xrightarrow{\underline{d}[k]} Q'}{P \mid Q \xrightarrow{\underline{c}[k]} P' \mid Q'}\ (*)$$

$$\textsf{rev res}\ \frac{P \xrightarrow{\underline{a}[k]} P'}{P\backslash L \xrightarrow{\underline{a}[k]} P'\backslash L}\ \underline{a} \notin L \qquad \textsf{rev con}\ \frac{P \xrightarrow{\underline{a}[k]} P'}{P \xrightarrow{\underline{a}[k]} S}\ S \stackrel{def}{=} P'$$

Fig. 3. Reverse SOS rules for CCB. The condition (*) is $\gamma(a,d) = c$, and $b \in \mathcal{WA}$.

in concert. The rule concert par requires that k is fresh in Q, correspondingly as in par. Moreover, we need to ensure that when we reverse h with the key l in P we do not leave out any actions with the key l in Q which make up a multiaction communication with the key l. Hence, we also include the premise $\textsf{fsh}[l](Q)$ in concert par. The rule concert act requires, correspondingly as act, that k is fresh in t. Our operational semantics guarantees that if a standard process evolves to $(t;b).P$, for some P, and P reverses an action with the key l, then l is fresh in t. Hence, we do not include $\textsf{fsh}[l](t)$ in the premises of concert act. Overall, the transitions in Figures 2–4 are labelled with $a[k] \in \mathcal{AK}$, or with $\underline{c}[l] \in \underline{\mathcal{A}}\mathcal{K}$, or with concerted actions $(a[k], \underline{c}[l])$.

Next, we recall the main new rewrite rules for a reduction relation for CCB in Figure 5. All the rules can be found in [15, 16] but here we only give rules for *promotion* of actions. These are prom, move-r, and move-l which promote weak bonds (here b) to strong bonds (here a). The rule prom applies to the full version of our prefix operator (with the ; construct), and move-r and move-l apply only to the simple prefix. These three rules are here to model what happens in chemical systems: a bond on a weak action is temporary and as soon as there is a strong action that can accommodate that bond (as the result of concerted actions) the bond establishes itself on the strong action thus releasing the weak action. In order to align the use of these three rules to what happens in chemical reactions, we insist that they are used as soon as they becomes applicable, a formal definition is given in [15, 16].

$$\textsf{aux1}\ \frac{\textsf{std}(P) \quad \textsf{fsh}[k](t)}{(t;b).P \xrightarrow{(b)[k]} (t;b[k]).P} \qquad \textsf{aux2}\ \frac{P \xrightarrow{(b)[k]} P' \quad \textsf{fsh}[k](t)}{(t;b').P \xrightarrow{(b)[k]} (t;b').P'}$$

$$\textsf{concert}\ \frac{P \xrightarrow{(b)[k]} P' \quad P' \xrightarrow{\underline{a}[l]} P'' \qquad Q \xrightarrow{\alpha[k]} Q' \quad Q' \xrightarrow{\underline{d}[l]} Q''}{P \mid Q \xrightarrow{\{e[k],\underline{f}[l]\}} P'' \mid Q''}(*)$$

$$\textsf{concert act}\ \frac{P \xrightarrow{\{a[k],\underline{h}[l]\}} P' \quad \textsf{fsh}[k](t)}{(t;b).P \xrightarrow{\{a[k],\underline{h}[l]\}} (t;b).P'}$$

$$\textsf{concert par}\ \frac{P \xrightarrow{\{a[k],\underline{h}[l]\}} P' \quad \textsf{fsh}[k](Q) \quad \textsf{fsh}[l](Q)}{P \mid Q \xrightarrow{\{a[k],\underline{h}[l]\}} P' \mid Q}$$

$$\textsf{concert res}\ \frac{P \xrightarrow{\{a[k],\underline{h}[l]\}} P'}{P\backslash L \xrightarrow{\{a[k],\underline{h}[l]\}} P'\backslash L}(**)$$

Fig. 4. SOS rules for concerted actions in CCB. The condition (*) is 1. $\alpha = c \vee \alpha = (c)$ and $\exists c \in \mathcal{A} | \gamma(b,c) = e$, and 2. $\gamma(a,d) = f$. The condition (**) is $a, \underline{h} \notin L \cup (L)$. Recall that $t \in \mathcal{A}\mathcal{K}^*$, and $b \in \mathcal{W}\mathcal{A}$.

$$\begin{aligned}
&\textsf{prom}: && (s,a,s';b[k]).P \Rightarrow (s,a[k],s';b).P && \text{if } a \in \mathcal{S}\mathcal{A}, b \in \mathcal{W}\mathcal{A}\\
&\textsf{move-r}: && (s,a,s',b[k],s'').P \Rightarrow (s,a[k],s',b,s'').P && \text{if } a \in \mathcal{S}\mathcal{A}, b \in \mathcal{W}\mathcal{A}\\
&\textsf{move-l}: && (s,b[k],s',a,s'').P \Rightarrow (s,b,s',a[k],s'').P && \text{if } a \in \mathcal{S}\mathcal{A}, b \in \mathcal{W}\mathcal{A}
\end{aligned}$$

Fig. 5. New reduction rules for CCB. Sequences s, s', s'' are members of $(\mathcal{A} \cup \mathcal{A}\mathcal{K})^*$.

We shall call henceforth the transitions derived by the forward SOS rules the *forward transitions* and, the transitions derived by the reverse SOS rules the *reverse transitions*. Correspondingly, there are the *concerted (action)* transitions.

The autoprotolysis of water in CCB When modelling the autoprotolysis of water in CCB, we shall model the hydrogen and oxygen atoms as processes H and O as follows, where h, o are actions representing the bonding capabilities of the atoms and n, p representing negative and positive charges, respectively. H' and O' are process constants, and p and n are weak actions.

$$H \stackrel{def}{=} (h;p).H' \qquad O \stackrel{def}{=} (o,o,n).O'$$

The synchronisation function γ is as follows:

$$\gamma(h,o) = ho \qquad \gamma(n,p) = np \qquad \gamma(n,h) = nh$$

Each water molecule is a structure consisting of two hydrogen atoms and one oxygen atom which are bonded appropriately. We shall use subscripts to distinguish the individual copies of atoms and actions; for example H_1 is a specific copy of hydrogen defined by $(h_1;p).H_1'$, similarly for O_1 defined as $(o_1,o_2,n).O_1'$. The atoms are composed with the parallel composition operator "|" using the

communication keys (which are natural numbers) to combine actions into bonds. So a water molecule is modelled by the following process, where the key 1 shows that h_1 of H_1 has bonded with o_1 of O_1 (correspondingly for key 2). The restriction $\backslash\{h_1, h_2, o_1, o_2\}$ ensures that these actions cannot happen on their own, but only together with their partners, forming a bond.

$$((h_1[1]; p).H_1' \mid (h_2[2]; p).H_2' \mid (o_1[1], o_2[2], n).O_1') \setminus \{h_1, h_2, o_1, o_2\}$$

The system of two water molecules in Figure 1 is represented by the parallel composition of two water processes, where the restriction $\backslash\{n, p\}$ represses actions n, p from taking place separately by forcing them to combine into bonds (according to γ).

$$\begin{aligned}&(((h_1[1]; p).H_1' \mid (h_2[2]; p).H_2' \mid (o_1[1], o_2[2], n).O_1') \setminus \{h_1, h_2, o_1, o_2\} \mid \\ &((h_3[3]; p).H_3' \mid (h_4[4]; p).H_4' \mid (o_3[3], o_4[4], n).O_2') \setminus \{h_3, h_4, o_3, o_4\}) \setminus \{n, p\}\end{aligned}$$

Following a general principle in process calculi in the style of CCB we can move the restrictions to the outside. The rule used can be written as $(P \mid Q) \setminus L = P \setminus L \mid Q$ if the actions of L are not used in Q. Applying this gives us a water molecule modelled as follows:

$$\begin{aligned}&((h_1[1]; p).H_1' \mid (h_2[2]; p).H_2' \mid (o_1[1], o_2[2], n).O_1') \mid (h_3[3]; p).H_3' \mid \\ &(h_4[4]; p).H_4' \mid (o_3[3], o_4[4], n).O_2')) \setminus \{h_1, h_2, o_1, o_2\} \setminus \{h_3, h_4, o_3, o_4\} \setminus \{n, p\}\end{aligned}$$

Note the $\underline{h_i}$, $\underline{o_j}$, and $\underline{n}$ are not restricted: this allows us to break bonds via concerted actions involving these actions. We will see an example of this shortly. We now leave out the restrictions to improve readability.

Actions n in O_1 and p in H_3 combine (we use the new key 5), representing a transfer of a proton from one atom of oxygen (O_2 in our model) to another one (O_1 in our model). As a hydrogen atom consists of a proton and an electron, and the electron stays in such a transfer, it can either be called a proton transfer or the transfer of a (positively charged) hydrogen atom. We perform the transfer of H_3 from O_2 to O_1. The creation of the bond with key 5 from O_1 to H_3 forces a break of the bond with key 3 (between h_3 and o_3) due to the property of the operator $(s; b).P$ discussed earlier. These two reactions happen almost simultaneously so we represent them as a pair of *concerted actions*.

$$\begin{aligned}&(h_1[1]; p).H_1' \mid (h_2[2]; p).H_2' \mid (o_1[1], o_2[2], n).O_1' \mid (h_3[3]; p).H_3' \\ &\mid (h_4[4]; p).H_4' \mid (o_3[3], o_4[4], n).O_2') \\ &\xrightarrow{\{np[5], \underline{h_3 o_3}[3]\}} \\ &((h_1[1]; p).H_1' \mid (h_2[2]; p).H_2' \mid (o_1[1], o_2[2], n\mathbf{[5]}).O_1' \mid (\boldsymbol{h_3}; p\mathbf{[5]}).H_3' \\ &\mid (h_4[4]; p).H_4' \mid (\boldsymbol{o_3}, o_4[4], n).O_2'\end{aligned}$$

We have now arrived at the state on the right hand side in Figure 1. There are weak bonds between n and p (denoted by key 5) and *strong* bonds between h_i

and o_j for all appropriate i, j. Since H_3 is weakly bonded to O_1 and its strong capability h_3 has become available, the bond 5 gets promoted to the stronger bond, releasing the capability p of H_3. We represent this change as a rewrite and we obtain the following process:

$$
\begin{array}{l}
((h_1[1];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1[1], o_2[2], n\mathbf{[5]}).O_1' \mid (\boldsymbol{h_3};p\mathbf{[5]}).H_3' \\
\mid (h_4[4];p).H_4' \mid (\boldsymbol{o_3}, o_4[4], n).O_2' \\
\Rightarrow \\
((h_1[1];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1[1], o_2[2], n[5]).O_1') \mid (h_3\mathbf{[5]};p).H_3') \\
\mid (h_4[4];p).H_4' \mid (o_3, o_4[4], n).O_2'
\end{array}
$$

Note that we wrote h_3, o_3 and the key 5, the actions and keys affected by the promotion, in bold font to improve readability. We shall do correspondingly below.

Oxygen O_1 is still blocked, which represents it being fully bonded (and positively charged). Oxygen O_2 has a free n capability and can remove any of the hydrogens from O_1. As a result the process can reverse to its original state.

We show this by again transferring H_3. We then execute promotion again:

$$
\begin{array}{l}
(((h_1[1];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1[1], o_2[2], n[5]).O_1') \mid (h_3[5];p).H_3') \\
\mid (h_4[4];p).H_4' \mid (o_3, o_4[4], n).O_2' \\
\xrightarrow{\{np[3], \underline{nh_3}[5]\}} \\
(((h_1[1];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1[1], o_2[2], \boldsymbol{n}).O_1') \mid (\boldsymbol{h_3};\boldsymbol{p}\mathbf{[3]}).H_3') \\
\mid (h_4[4];p).H_4' \mid (o_3, o_4[4], \boldsymbol{n}\mathbf{[3]}).O_2' \\
\Rightarrow \\
(((h_1[1];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1[1], o_2[2], \boldsymbol{n}).O_1') \mid (\boldsymbol{h_3}\mathbf{[3]};\boldsymbol{p}).H_3') \\
\mid (h_4[4];p).H_4' \mid (\boldsymbol{o_3}\mathbf{[3]}, o_4[4], \boldsymbol{n}).O_2'
\end{array}
$$

This corresponds to the original process. Putting back restrictions we obtain

$$
\begin{array}{l}
((h_1[1];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1[1], o_2[2], n).O_1' \mid (h_3[3];p).H_3' \\
\mid (h_4[4];p).H_4' \mid (o_3[3], o_4[4], n).O_2') \setminus \{h_1, h_2, o_1, o_2\} \setminus \{h_3, h_4, o_3, o_4\} \setminus \{n, p\}
\end{array}
$$

and then if we apply the movement of restrictions in reverse we get

$$
\begin{array}{l}
(((h_1[1];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1[1], o_2[2], n).O_1') \setminus \{h_1, h_2, o_1, o_2\} \mid \\
((h_3[3];p).H_3' \mid (h_4[4];p).H_4' \mid (o_3[3], o_4[4], n).O_2') \setminus \{h_3, h_4, o_3, o_4\}) \setminus \{n, p\}
\end{array}
$$

3.2 Bonding Calculus

In this subsection we recall briefly the Bonding Calculus [1], and illustrate its expressiveness by modelling the autoprotolysis of water.

Definition of the Bonding Calculus The abstraction "processes as interactions" from process calculi is used in the Bonding Calculus, but processes are not able to communicate values in order to interact. Just like in the BNGL [9], the Bonding Calculus allows the use of molecule sites having the same name, while this is not possible in the κ-calculus. While the κ-calculus describes molecules as a set of sites and uses rules to manipulate these sites between two or more molecules, in the Bonding Calculus a molecule is described by the sequence of operations it can perform on its sites (including also non-deterministic choices), regardless of the form of the other molecules. This allows to use the compositionality of the process calculus.

The syntax of the Bonding Calculus syntax is presented in Figure 6. Let us consider the set $\mathbb{N}$ of natural numbers, the set $\mathcal{N} = \{x, x^+, x^-, \dots\}$ of bond names, the set $\mathcal{M} = \{a, b, \dots\}$ of molecules and the set $\mathcal{P} = \{P, Q, \dots\}$ of processes. A multiset over $\mathcal{N}$ is defined as a partial function $N : \mathcal{N} \to \mathbb{N}$. In the Bonding Calculus each molecule has a unique name, and the bond x between two molecules a and b is denoted by $\{a -^x b\}$.

Bonds	L	$::=$	$\emptyset$	(empty)
		$\mid$	$\{a -^x b\}$	(bond)
		$\mid$	$L \uplus L$	(union)
Actions	α	$::=$	$\overline{x}(b)$	(bond)
		$\mid$	$\underline{x}(b)$	(unbond)
Processes	P	$::=$	$\mathbf{0}$	(empty)
		$\mid$	$\alpha.P$	(action prefix)
		$\mid$	$P + Q$	(choice)
		$\mid$	$P \mid Q$	(parallel)
		$\mid$	**if** $a \eqdot^L b$ **then** P **else** Q	(testing)
		$\mid$	$A(b_1, \dots, b_n)$	(identifier)
Definition	$A(a_1, \dots, a_n)$	$::=$	P	(recursion)
System	S	$::=$	$P \mid\mid L$.	

Fig. 6. Syntax of the Bonding Calculus

A bond prefix $\overline{x}(b)$ is used to indicate the availability of a molecule with name b to create a new bond with name x, while an unbond prefix $\underline{x}(b)$ indicates the availability of b to destroy an existing bond x. Creating or breaking a bond leads to an update of the global bond memory L. As several similar bonds can exist between the same molecules, L is actually a multiset of bonds.

The process $\mathbf{0}$ denotes inactivity. The availability to perform an action α, and then to continue the execution as process P is denoted by the process $\alpha.P$. The process $P + Q$ offers a choice between the processes P and Q, while the process $P \mid Q$ allows the execution of processes P and Q in parallel, with possible interactions between them by using appropriate actions.

As we work with bonds, we use the function $\fallingdotseq$: $\mathcal{M} \times \mathbb{N}^{\mathcal{N}} \times \mathcal{M} \to \mathit{Bool}$ to check whether between two molecules there exist certain bonds. For example, $a \fallingdotseq^N b$ checks for the existence of all bonds in N between the molecules a and b; it returns true when such bonds exist, and false otherwise. When we consider $N = \emptyset$, then $a \fallingdotseq^\emptyset b$ checks if at least a bond exists between the two molecules. When $b = \varepsilon$, then $a \fallingdotseq^N \varepsilon$ checks if a has all of bonds from N, regardless of the molecules he has them with. The Boolean result $a \fallingdotseq^N b$ used in the testing process is defined formally as:

$$a \fallingdotseq^N b = \begin{cases} (\biguplus_{x \in N} \{a -^x b\}) \in L & N \neq \emptyset \text{ and } a \neq b \neq \varepsilon \\ (\biguplus_{x \in \mathcal{N}} \{a -^x b\}) \cap L \neq \emptyset & N = \emptyset \text{ and } a \neq b \neq \varepsilon \\ \bigwedge_{x \in N} (|L|_{a,x} = |N|_x) & N \neq \emptyset \text{ and } a \neq \varepsilon \text{ and } b = \varepsilon \\ \mathit{undefined} & \mathit{otherwise}, \end{cases}$$

where $|L|_{a,x}$ is the number of bonds containing the molecule a and bond name x that appear in the multiset L, while $|N|_x$ is the number of occurrences of x in N.

Depending on the truth value of $a \fallingdotseq^N b$, the process **if** $a \fallingdotseq^N b$ **then** P **else** Q executes either P or Q. An identifier $A(b_1, \ldots, b_n)$ is used to provide recursion by creating new instances of processes defined as $A(a_1, \ldots, a_n) = P$, where $a_i \neq a_j$ for all $i \neq j \in \{1, \ldots, n\}$; the new process is defined as $A(b_1, \ldots, b_n) = P\{b_1/a_1, \ldots, b_n/a_n\}$, where $\{b_i/a_i\}$ denotes the replacement of variable a_i by value b_i. A system S is given as a composition of a process P and the multiset of bonds L, written as $P \mid\mid L$.

The structural congruence relation $\equiv$ is the least congruence such that $(\mathcal{P}, +, \mathbf{0})$ and $(\mathcal{P}, |, \mathbf{0})$ are commutative monoids and the unfolding law $A(b_1, \ldots, b_n) \equiv P\{b_1/a_1, \ldots, b_n/a_n\}$ holds whenever $A(a_1, \ldots, a_n) = P$.

The calculus presented in [1] was intended to model the creation and breaking of covalent bonds. In order to be able to model both covalent and hydrogen bonds, we apply a minor update to the operational semantics in [1] because we need two instances of the rules used to create and to break bonds. The only difference between the two instances of the same rule is given by the names of the bonds appearing in the interacting processes, and by the fact that a bond cannot be created using the names x^+ and x^- if other bonds exist between the same molecules; more details about this restriction are given in the example below.

The operational semantics of the Bonding Calculus is given in Figure 7. The rules (CREATE1) and (CREATE2) describe the creation of a new bond $\{a -^x b\}$, while the rules (REMOVE1) and (REMOVE2) describe the breaking of a bond $\{a -^x b\}$. If there exist two bonds $\{a -^x b\}$ in L, then any of these bonds is broken. The rule (PAR) is used to compose processes in parallel, while the rules (TRUE) and (FALSE) choose one of the branches of the testing process based on the result of the checking. The rule (IDE) describes the recursion, while the (STRUCT) rule indicates the fact that we reason up to the structural congruence.

$$\textsf{(CREATE1)}\ \frac{P = \overline{x}(b).P' + P'' \quad Q = \overline{x}(a).Q' + Q''}{(P \mid Q) \mid\mid L \to (P' \mid Q') \mid\mid L \uplus \{a -^{x} b\}}$$

$$\textsf{(CREATE2)}\ \frac{P = \overline{x^{+}}(b).P' + P'' \quad Q = \overline{x^{-}}(a).Q' + Q'' \quad a \leftrightarrows^{\emptyset} b \ is\ false\ w.r.t.\ L}{(P \mid Q) \mid\mid L \to (P' \mid Q') \mid\mid L \uplus \{a -^{x} b\}}$$

$$\textsf{(REMOVE1)}\ \frac{P = \underline{x}(b).P' + P'' \quad Q = \underline{x}(a).Q' + Q'' \quad a \leftrightarrows^{x} b \ is\ true\ w.r.t.\ L}{(P \mid Q) \mid\mid L \to (P' \mid Q') \mid\mid L \backslash \{a -^{x} b\}}$$

$$\textsf{(REMOVE2)}\ \frac{P = \underline{x^{+}}(b).P' + P'' \quad Q = \underline{x^{-}}(a).Q' + Q'' \quad a \leftrightarrows^{N} b \ is\ true\ w.r.t.\ L}{(P \mid Q) \mid\mid L \to (P' \mid Q') \mid\mid L \backslash \{a -^{x} b\}}$$

$$\textsf{(PAR)}\ \frac{P \mid\mid L \to P' \mid\mid L}{(P \mid Q) \mid\mid L \to (P' \mid Q) \mid\mid L}$$

$$\textsf{(TRUE)}\ \frac{a \leftrightarrows^{N} b \ is\ true\ w.r.t.\ L}{(\textbf{if}\ a \leftrightarrows^{N} b\ \textbf{then}\ P\ \textbf{else}\ Q) \mid\mid L \to P \mid\mid L}$$

$$\textsf{(FALSE)}\ \frac{a \leftrightarrows^{N} b \ is\ false\ w.r.t.\ L}{(\textbf{if}\ a \leftrightarrows^{N} b\ \textbf{then}\ P\ \textbf{else}\ Q) \mid\mid L \to Q \mid\mid L}$$

$$\textsf{(IDE)}\ \frac{P\{b_1/a_1, \ldots, b_n/a_n\} \to P'}{A(b_1, \ldots, b_n) \to P'} \ \text{ if } \ A(a_1, \ldots, a_n) = P$$

$$\textsf{(STRUCT)}\ \frac{S_1 \to S_1' \quad S_1 \equiv S_2 \quad S_2 \to S_2'}{S_1' \to S_2'}$$

Fig. 7. Operational Semantics of the Bonding Calculus.

The autoprotolysis of water in the Bonding Calculus We use two types of bond names, namely c and h, to stand for the covalent and hydrogen bonds, respectively. Using our calculus, the system composed of two molecules of water is described by:

$$MolOxy_2(O_1) \mid MolHy_1(H_1) \mid MolHy_1(H_2)$$

$$\mid MolOxy_2(O_2) \mid MolHy_1(H_3) \mid MolHy_1(H_4)$$

$$\mid\mid \{O_1 -^{c} H_1, O_1 -^{c} H_2, O_2 -^{c} H_3, O_2 -^{c} H_4\}$$

where the molecules are those of hydrogen and oxygen that are described below:

$$MolHy_0(H_i) = \overline{c}(H_i).MolHy_1(H_i)$$
$$MolHy_1(H_i) = \underline{c}(H_i).MolHy_0(H_i) + \overline{h^{+}}(H_i).MolHy_2(H_i);$$
$$MolHy_2(H_i) = \underline{c}(H_i).\overline{c}(H_i).\underline{h^{+}}(H_i).MolHy_1(H_i).$$
$$MolOxy_0(O_i) = \overline{c}(O_i).MolOxy_1(O_i);$$
$$MolOxy_1(O_i) = \underline{c}(O_i).MolOxy_0(O_i) + \overline{c}(O_i).MolOxy_2(O_i);$$
$$MolOxy_2(O_i) = \underline{c}(O_i).MolOxy_1(O_i) + \overline{h^{-}}(O_i).MolOxy_3(O_i).$$
$$MolOxy_3(O_i) = \underline{h^{-}}(O_i).MolOxy_2(O_i).$$

Each molecule of water is a structure consisting of one molecule of oxygen and two molecules of hydrogen which are properly bonded. For example, the process $MolOxy_2(O_1) \mid MolHy_1(H_1) \mid MolHy(H_2)$ together with the bonds $\{O_1 -^c H_1, O_1 -^c H_2\}$ model one molecule of water. We use unique names for the molecules given as O_i (for oxygen) and H_i (for hydrogen), while the processes having the names $MolHy_i$ and $MolOxy_i$ identify processes modelling hydrogen and oxygen molecules with i bonds, respectively. For example, the process $MolOxy_1(O_i)$ can either create or break bonds, and this is why we use the operator $+$ to describe such a (non-deterministic) choice.

Now we present the steps of one of the possible sequences of reactions modelling the autoprotolysis of water. The system of two molecules of water can be rewritten as follows (where we extend the definitions for the processes that will interact in the next step, and bold the actions to be executed):

$$\underline{c}(O_1).MolOxy_1(O_1) + \overline{\mathbf{h}^-}(\mathbf{O_1}).MolOxy_3(O_1) \mid MolHy_1(H_1) \mid MolHy_1(H_2)$$

$$\mid MolOxy_2(O_2) \mid MolHy_1(H_3) \mid \underline{c}(H_4).MolHy_0(H_4) + \overline{\mathbf{h}^+}(\mathbf{H_4}).MolHy_2(H_4)$$

$$\mid\mid \{O_1 -^c H_1, O_1 -^c H_2, O_2 -^c H_3, O_2 -^c H_4\}$$

This leads to the next system, where we again bold the processes to be executed:

$$MolOxy_3(O_1) \mid MolHy_1(H_1) \mid MolHy_1(H_2)$$

$$\mid \underline{\mathbf{c}}(\mathbf{O_2}).MolOxy_1(O_2) + \overline{h^-}(O_2).MolOxy_3(O_1) \mid MolHy_1(H_3)$$

$$\mid \underline{\mathbf{c}}(\mathbf{H_4}).\overline{c}(H_4).\underline{h^+}(H_4).MolHy_1(H_4)$$

$$\mid\mid \{O_1 -^c H_1, O_1 -^c H_2, O_1 -^h H_4, O_2 -^c H_3, O_2 -^c H_4\}$$

The creation of the hydrogen bond forces the break of the other bond in which the hydrogen molecule H_4 is involved. This leads to the following system containing the H_3O and HO molecules:

$$MolOxy_3(O_1) \mid MolHy_1(H_1) \mid MolHy_1(H_2)$$

$$\mid \underline{c}(O_2).MolOxy_0(O_2) + \overline{\mathbf{c}}(\mathbf{O_2}).MolOxy_2(O_2)$$

$$\mid MolHy_1(H_3) \mid \overline{\mathbf{c}}(\mathbf{H_4}).\underline{h^+}(H_4).MolHy_1(H_4)$$

$$\mid\mid \{O_1 -^c H_1, O_1 -^c H_2, O_1 -^h H_4, O_2 -^c H_3\}$$

Since some bonds are weaker, the system is evolving to:

$$\underline{\mathbf{h}^-}(\mathbf{O_1}).MolOxy_2(O_1) \mid MolHy_1(H_1) \mid MolHy_1(H_2)$$

$$\mid MolOxy_2(O_2) \mid MolHy_1(H_3) \mid \underline{\mathbf{h}^+}(\mathbf{H_4}).MolHy_1(H_4)$$

$$\mid\mid \{O_1 -^c H_1, O_1 -^c H_2, O_1 -^h H_4, O_2 -^c H_3, O_2 -^c H_4\}$$

followed by the breaking of the hydrogen bond $O_1 -^h H_4$:

$$MolOxy_2(O_1) \mid MolHy_1(H_1) \mid MolHy_1(H_2)$$
$$\mid MolOxy_2(O_2) \mid MolHy_1(H_3) \mid MolHy_1(H_4)$$
$$\| \{O_1 -^c H_1, O_1 -^c H_2, O_2 -^c H_3, O_2 -^c H_4\}$$

The obtained system contains again two water molecules of water.

3.3 Reversing Petri Nets

In this subsection we present Reversing Petri Nets [23] (RPNs, pronounced as 'reversing Petri nets'), an extension of Petri nets developed for the modelling reversing computations, and we employ the formalism to model the autoprotolysis of water.

Definition of RPNs We consider an extension of reversing Petri nets suitable for describing chemical reactions by allowing multiple tokens of the same type as well as the possibility for transitions to break bonds. Thus, a transition may simultaneously create and/or destroy bonds, and its reversal results in the opposite effect. Formally, a Reversing Petri net is defined as follows:

Definition 1. A *reversing Petri net* (RPN) is a tuple (P, T, A, A_V, B, F) where:

1. P is a finite set of *places* and T is a finite set of *transitions*.
2. A is a finite set of *base* or *token types* ranged over by $a, b, \ldots$. $\overline{A} = \{\overline{a} \mid a \in A\}$ contains a "negative" version for each token type. We assume that for any token type a there may exist a finite number of *token instances*. We write $a_1, \ldots,$ for instances of type a and A_I for the set of all token instances.
3. A_V is a finite set of *token variables*. We write *type*(v) for the type of variable v and assume that *type*$(v) \in A$ for all $v \in A_V$.
4. $B \subseteq A \times A$ is a finite set of undirected *bond* types ranged over by $\beta, \gamma, \ldots$. We use the notation $a-b$ for a bond $(a, b) \in B$. $\overline{B} = \{\overline{\beta} \mid \beta \in B\}$ contains a "negative" version for each bond type. $B_I \subseteq A_I \times A_I$ is a finite set of *bond instances*, where we write β_i for elements of B.
5. $F : (P \times T \cup T \times P) \to \mathcal{P}(A_V \cup (A_V \times A_V) \cup \overline{A} \cup \overline{B})$ is a set of directed labelled *arcs*.

A reversing Petri net is built on the basis of a set of *tokens* or *bases*. These are organised in a set of token types A, where each token type is associated with a set of token instances. Token instances correspond to the basic entities that occur in a system and they may occur as stand-alone elements but as computation proceeds they may also merge together to form *bond instances*. Places and transitions have the standard meaning and are connected via directed arcs, which are labelled by a set of elements from $A_V \cup (A_V \times A_V) \cup \overline{A} \cup \overline{B}$. Intuitively, these labels express the requirements for a transition to fire when placed

on arcs incoming the transition, and the effects of the transition when placed on the outgoing arcs. Graphically, a RPN is portrayed as a directed bipartite graph where token instances are indicated by •, places by circles, transitions by boxes, and bond instances by lines between token instances.

Before we recall the semantics of RPNs we need to introduce some notation. Note that in what follows we omit the discussion of negative tokens and negative bonds as they are not relevant to our case study. We write $\circ t = \{x \in P \mid F(x,t) \neq \emptyset\}$ and $t\circ = \{x \in P \mid F(t,x) \neq \emptyset\}$ for the incoming and outgoing places of transition t, respectively. Furthermore, we write $\mathsf{pre}(t) = \bigcup_{x \in P} F(x,t)$ for the union of all labels on the incoming arcs of transition t, and $\mathsf{post}(t) = \bigcup_{x \in P} F(t,x)$ for the union of all labels on the outgoing arcs of transition t.

Definition 2. A reversing Petri net is *well-formed*, if for all $t \in T$:

1. $A_V \cap \mathsf{pre}(t) = A_V \cap \mathsf{post}(t)$,
2. $F(t,x) \cap F(t,y) \cap A_V = \emptyset$ for all $x, y \in P$, $x \neq y$.

Thus, a reversing Petri net is well-formed if (1) whenever a variable exists in the incoming arcs of a transition then it also exists on the outgoing arcs, which implies that transitions do not erase tokens, and (2) tokens/bonds cannot be cloned into more than one outgoing places.

As with standard Petri nets the association of token/bond instances to places is called a *marking* such that $M : P \to 2^{A_I \cup B_I}$, where we assume that if $(u,v) \in M(x)$ then $u, v \in M(x)$. In addition, we employ the notion of a *history*, which assigns a memory to each transition $H : T \to \mathbb{N}$. Intuitively, a history of $H(t) = 0$ for some $t \in T$ captures that the transition has not taken place, or every execution of it has been reversed, and a history of $H(t) = k, k > 0$, captures that the transition had k forward executions that have not been reversed. Note that $H(t) > 1$ may arise due to the consecutive execution of the transition with different token instances. A pair of a marking and a history, $\langle M, H \rangle$, describes a *state* of a RPN with $\langle M_0, H_0 \rangle$ the initial state, where $H_0(t) = 0$ for all $t \in T$.

Finally, we define $\mathsf{con}(a_i, C)$, where $a_i \in A_I$ and $C \subseteq 2^{A_I \cup B_I}$, to be the token instances connected to a_i as well as the bonds creating these connections according to set C.

Forward Execution. During the forward execution of a transition in a RPN, a set of tokens and bonds, as specified by the incoming arcs of the transition, are selected and moved to the outgoing places of the transition, as specified by the transition's outgoing arcs, possibly forming or destructing bonds, as necessary. Due to the presence of multiple instances of the same token type, it is possible that different token instances are selected during the transition's execution.

A transition is forward-enabled in a state $\langle M, H \rangle$ of a reversing Petri net if there exists a selection of token instances available at the incoming places of the transition matching the requirements on the transitions incoming arcs. Formally:

Definition 3. Given a RPN (P, T, A, A_V, B, F), a state $\langle M, H \rangle$, and a transition t, we say that t is *forward-enabled* in $\langle M, H \rangle$ if there exists a surjective function $U : \mathsf{pre}(t) \cap A_V \to A_I$ such that:

1. for all $v \in \mathsf{pre}(t)$, if $\mathit{type}(v) = a$ then $\mathit{type}(U(v)) = a$
2. for all $a \in F(x,t)$, then $U(a) \in M(x)$ and for all $(a,b) \in F(x,t)$, then $(U(a), U(b)) \in M(x)$,
3. for all $(a,b) \in \mathsf{post}(t) - \mathsf{pre}(t)$ then $(U(a), U(b)) \notin M(x)$ for all $x \in {\circ}t$.

Thus, t is enabled in state $\langle M, H\rangle$ if (1) there is a type-respecting assignment of token instances to the variables on the incoming edges, with (2) the token instances originating from the appropriate input places of the transition and connected with bonds as required by the variable bonds occurring on the incoming edges, and (3) if a bond occurs in the outgoing edges of the transition but not the incoming ones, then the selected instances associated with the bond's variables should not be bonded together in the incoming places of the transition (thus transitions do not recreate bonds). We refer to U as a forward enabling assignment.

To execute a transition t according to an enabling assignment U, the selected token instances, along with their connected components, are relocated to the outgoing places of the transition as specified by the outgoing arcs, with bonds created and destructed accordingly. Furthermore, the history of the executed transition is increased by one.

Definition 4. Given a RPN (P, T, A, A_V, B, F), a state $\langle M, H\rangle$, and an enabling assignment U, we write $\langle M, H\rangle \xrightarrow{t}_{\mathrm{S}} \langle M', H'\rangle$ where for all $x \in P$:

$$M'(x) = M(x) - \bigcup_{a \in f(x,t)} \mathsf{con}(U(a), M(x)) \cup \bigcup_{a \in f(t,x), U(a) \in M(y)} \mathsf{con}(U(a), S)$$

where $S = (M(y) - \{(U(a), U(b)) \mid (a,b) \in F(y,t)\}) \cup \{(U(a), U(b)) \mid (a,b) \in F(t,x)\}$

$$\text{and} \qquad H'(t') = \begin{cases} H(t') + 1, & \text{if } t' = t \\ H(t'), & \text{otherwise} \end{cases}$$

Reversing Execution. We now move on to reversing transitions. A transition can be reversed in a certain state if it has been previously executed and there exist token instances in its output places that match the requirements on its outgoing arcs. Specifically, we define the notion of reverse enabledness as follows:

Definition 5. Consider a RPN (P, T, A, A_V, B, F), a state $\langle M, H\rangle$, and a transition t. We say that t is *reverse-enabled* in $\langle M, H\rangle$ if (1) $H(t) \neq 0$, and (2) there exists a surjective function $W : \mathsf{post}(t) \cap A_V \to A_I$ such that:

1. for all $v \in \mathsf{post}(t)$, if $\mathit{type}(v) = a$ then $\mathit{type}(W(v)) = a$,
2. for all $a \in F(t,x)$, then $W(a) \in M(x)$ and for all $(a,b) \in F(t,x)$, then $(W(a), W(b)) \in M(x)$,
3. for all $(a,b) \in \mathsf{pre}(t) - \mathsf{post}(t)$ then $(W(a), W(b)) \notin M(x)$ for all $x \in {\circ}t$.

Thus, a transition t is reverse-enabled in $\langle M, H\rangle$ if (1) the transition has been executed and (2) there exists a type-respecting assignment of token instances, from the instances in the out-places of the transition, to the variables on the

outgoing edges of the transition, and where the instances are connected with bonds as required by the transition's outgoing edges. Also we do not recreate existing bonds when going backwards. We refer to W as a reversal enabling assignment. To implement the reversal of a transition t according to a reversal enabling assignment W, the selected instances are relocated from the outgoing places of the transition to the incoming places, as specified by the incoming arcs of the transition, with bonds created and destructed accordingly.

Definition 6. Given a RPN (P, T, A, A_V, B, F), a state $\langle M, H\rangle$, and a transition t reverse-enabled in $\langle M, H\rangle$ with W a reversal enabling assignment, we write $\langle M, H\rangle \overset{t}{\rightsquigarrow} \langle M', H'\rangle$ where for all x:

$$M'(x) = M(x) - \bigcup_{a \in f(t,x)} \mathsf{con}(W(a), M(x)) \cup \bigcup_{a \in f(x,t), W(a) \in M(y)} \mathsf{con}(W(a), S)$$

where $S = (M(y) - \{(W(a), W(b)) \mid (a, b) \in F(t, y)\}) \cup \{(W(a), W(b)) \mid (a, b) \in F(x, t)\}$

$$\text{and} \qquad H'(t') = \begin{cases} H(t') - 1, & \text{if } t' = t \\ H(t'), & \text{otherwise} \end{cases}$$

The autoprotolysis of water in RPNs Figure 8 shows the graphical representation of the forming of a water molecule as a RPN. In this model, we assume two token types, H for hydrogen and O for oxygen. They are instantiated via four token instances of H (H_1, H_2, H_3, and H_4) and two token instances of O, (O_1 and O_2). The net consists of five places and three transitions and the edges between them are associated with token variables and bonds, where we assume that $\mathit{type}(o) = \mathit{type}(o_1) = \mathit{type}(o_2) = O$ and $\mathit{type}(h) = \mathit{type}(h_1) = \mathit{type}(h_2) = \mathit{type}(h_3) = \mathit{type}(h_4) = H$. Looking at the transitions, transition t_1 models the formation of a bond between a hydrogen token and an oxygen token. Precisely, the transition stipulates a selection of two such molecules with the use of variables o and h on the incoming arcs of the transition which are bonded together, as described in the outgoing arc of the transition. Subsequently, transition t_2 completes the formation of a water molecule by selecting an oxygen token from place x and a hydrogen token from place v and forming a bond between them, placing the resulting component at place y. Note that the selected oxygen instance in this transition will be connected to a hydrogen token via a bond created by transition t_1; this bond is preserved and the component resulting from the creation of the new $o - h$ bond will be transferred to place y. Finally, transition t_3 models the autoprotolysis reaction: assuming the existence of two distinct oxygen instances, as required by the variables o_1 and o_2 on the incoming arc of the transition, connected with hydrogen instances as specified in $F(y, t_3)$, the transition breaks the bond $o_2 - h_3$ and forms the bond $o_1 - h_3$. As such, assuming the existence of two water molecules at place y, the transition will form a hydronium (H_3^+O) and a hydroxin (OH^-) molecule in place z of the net. The reversibility semantics of RPNs ensures that reversing the transition t_3 will result in the re-creation of two water molecules placed at y, while the use of variables

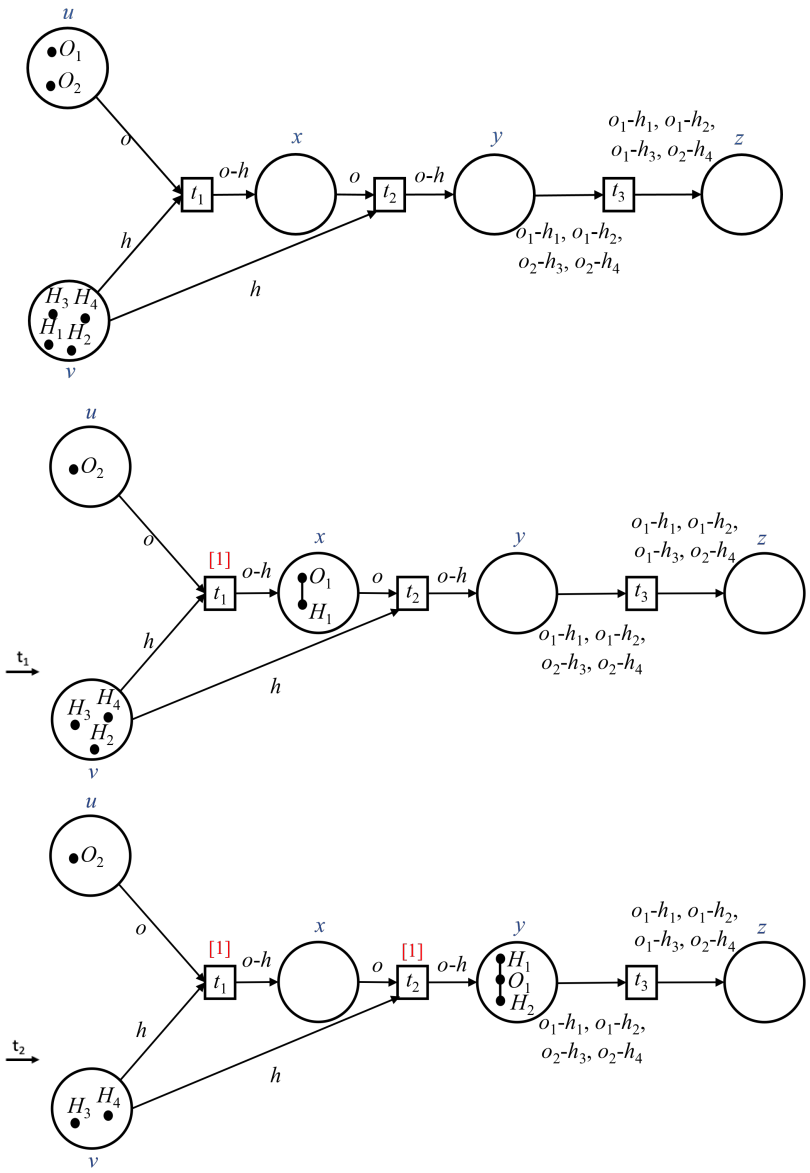

Fig. 8. RPN model of the formation of a water molecule.

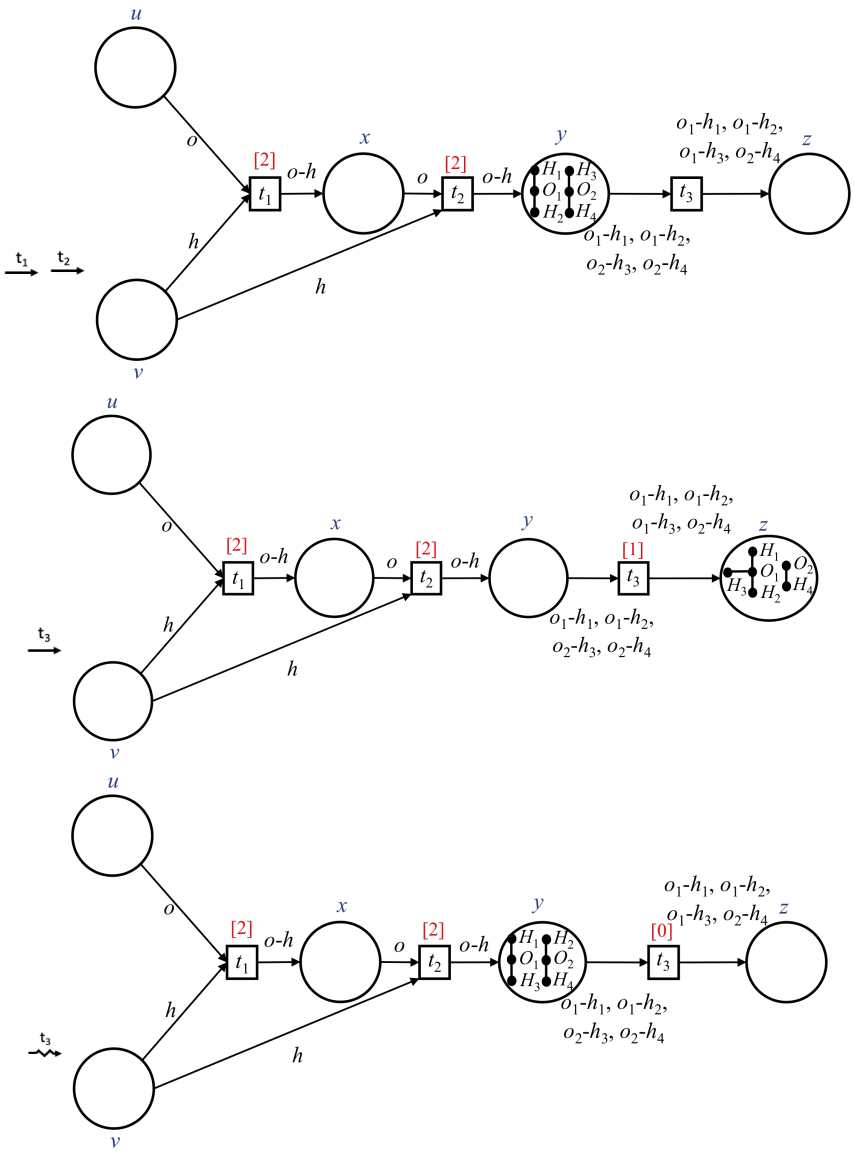

Fig. 9. RPN model of the execution of the autoprotolysis of water.

allows the formation of water molecules consisting of different bonds between the hydrogen and oxygen instances.

The first net in Figure 9 shows the system after the execution of transition t_1 with enabling assignment $U(h) = H_1$, $U(o) = O_1$. Note that the term [1] written over transition t_1 captures that at this point $H(t_1) = 1$ since the transition has been executed once. This notation is generally used for histories in the graphical representation with occasional missing histories corresponding to histories equal to 0. Subsequently, we have the model after execution of transition t_2 with enabling assignment $U(h) = H_2$, $U(o) = O_1$, creating the bond $O_1 - H_2$, thus forming the first water molecule. A second execution of transitions t_1 and t_2 results in the second molecule of water in the system, placed again at place y, as shown in the third net in the figure. At this state, transition t_3 is forward-enabled and, with enabling assignment $U(o_1) = O_1, U(o_2) = O_2, U(h_1) = H_1, U(h_2) = H_2, U(h_3) = H_3, U(h_4) = H_4$, we have the creation of the hydronium and hydroxide depicted at place z in the fourth net of the figure. At this stage, transition t_3 is now reverse-enabled and the last net in the figure illustrates the state resulting after reversing t_3 with reversal enabling assignment $W(o_1) = O_1, W(o_2) = O_2, W(h_1) = H_1, W(h_2) = H_3, W(h_3) = H_2, W(h_4) = H_4$.

4 Evaluation

We have presented three formalisms which can be used to model chemical reactions. CCB is a reversible version of ACP that employs communication keys to record executed actions. Its main feature is a mechanism to link forming and breaking of bonds, which gives rise to a type of explicit reversibility we call "locally controlled reversibility". We have modelled a simple covalent chemical reaction in CCB. A similar modelling approach can be used to model more complex atoms and reactions, for example, involving carbon atoms [16]. Finally, CCB can also be used to model reactions beyond simple chemical reactions [14]. In CCB, we can actually distinguish different instances of the same atom or molecule, and of identical actions in a process via the use of subscripts. As mentioned above, the reverse reaction in the autoprotolysis of water can work by transferring any of the hydrogens of the hydronium. When reversing the reaction in CCB, instead of the transition in Section 3.1, we could also have done this (writing the transition and the rewrite together):

$$
\begin{aligned}
&(((h_1[1];p).H_1' \mid (h_2[2];p).H_2' \mid (o_1[1],o_2[2],n[5]).O_1') \mid (h_3[5];p).H_3') \\
&\mid (h_4[4];p).H_4' \mid (o_3,o_4[4],n).O_2' \\
&\xrightarrow{\{np[3],\underline{nh_1}[1]\}}\Rightarrow \\
&(((\boldsymbol{h_1[3]};p).H_1' \mid (h_2[2];p).H_2' \mid (\boldsymbol{o_1[5]},o_2[2],\boldsymbol{n}).O_1') \mid (h_3[5];p).H_3') \\
&\mid (h_4[4];p).H_4' \mid (\boldsymbol{o_3[3]},o_4[4],\boldsymbol{n}).O_2'
\end{aligned}
$$

The result is different from that in Section 3.1, but identical from a chemical point of view, since the hydrogens are all identical. On the other hand a technique

called isotopic labelling can be used to trace atoms by using different isotopes of, in this case hydrogen, confirming that the different options happen in reality. In CCB, we can trace the atoms as well as show which results are identical from a chemical point of view (see Section 6.5 of [16]).

The Bonding Calculus is suitable for modelling in a natural way the autoprotolysis of water by using only bond and unbond actions. Simulations by using a software platform can describe the dynamics of the bonding systems, and so it is possible to test the validity of some underlying assumptions. Also, we can verify various properties of the bonding compounds described by using the calculus.

Reversing Petri Nets are Petri net structures that assume tokens to be distinct and persistent. During the execution of transitions individual tokens can be bonded/unbonded with each other, and the creation/destruction of these bonds is considered to be the effect of a transition, whereas their destruction/creation is the effect of the transition's reversal. Reversing Petri Nets are a natural choice to model and analyse biochemical reaction systems, such as the autoprotolysis of water, which by nature has multi-party interactions, is inherently concurrent, and features reversible behaviour. In particular, the feature of token multiplicity and the use of variables allows to non-deterministically select different combinations of atoms of a particular element when creating molecules. Also the ability of transitions to break bonds allows to model concerted actions where, for example, a transition simultaneously destroys a water molecule and creates a hydronium whose reversal results in the opposite effect. Moreover, the collective token interpretation adopted in the framework, treating all tokens of the same type as equivalent, allows the reaction to reverse into two different water molecules than the original ones, i.e. using different instances of the atoms (as is possible in CCB). Note that the presented model abstracts away the positive/negative charge of the atoms and captures the existence of electrons by the enabledness of transitions. A model at a lower level of abstraction would be possible by introducing tokens to represent the electrons bonded to the associated atom tokens to illustrate the relevant charges.

The three formalisms presented can model our example fairly well but, as expected, there are some differences. In order to evaluate each formalism, we consider as first criterion if all chemically valid interactions between the compounds of the reaction can be represented well in our formalisms. CCB shows the linked forming and breaking of bonds. RPNs can also express these concerted actions, since a transition enables the simultaneous creation and destruction of bonds. In the Bonding Calculus, this link is not expressed. Each of the formalisms can perform the forward reaction using any of the hydrogens involved. CCB and RPNs can perform the reverse reaction by transferring arbitrary hydrogens, whereas the Bonding Calculus in the reverse reaction permits only the transfer of exactly those hydrogens that were used in the forward reaction. All models presented use subscripts and enable the tracking of atoms.

The other criterion for assessing the suitability of our formalisms for the modelling of chemical reactions is to ask if they enable in the produced model any transitions that actually do not occur in reality. Each formalism does not permit

a H_3O^+ molecule to be formed directly. CCB allows one reaction which is not realistic: If there are many water molecules and therefore several hydroxide and water molecules at the same time, it is possible that the remaining hydrogen is transferred from the hydroxide to a water. In reality, this is not possible since the hydroxide is strongly negatively charged and no hydrogen bond can form. Due to the nondeterministic behaviour of processes written with the '+' operator, such as those for hydrogen and oxygen in Subsection 3.2, the Bonding Calculus also presents the same problem. However, this is not the case for RPNs since, on the one hand, a transition's conditions make restrictions on the types of molecules that will participate in a transition firing or its reversal and, on the other hand, places impose a form of locality for molecules. For instance, in the autoprotolysis example, each place is the location of specific types of molecules, e.g., transition t_3 modelling the autoprotolysis reaction is only applied on water molecules and its reversal only on pairs of a hydronium and a hydroxide molecule, as required.

There are a number of software tools that can aid simulation and analysis for our formalisms. Regarding the Bonding Calculus, we can simulate various bonding descriptions by using an existing software platform called UPPAAL (as shown in [1]). For CCB, there is a simulation tool presented in [14]. It allows a much closer form of representation of chemical notation than that possible with a typical programming language. Reversing Petri nets have been shown to be closely related to Coloured Petri Nets, as a subset of the former model has been encoded into the latter [3]. Thus, an algorithmic translation can be implemented that transforms RPNs to CPNs in an automated manner using the transformation techniques discussed in [3]. This allows RPNs to exploit tools such as CPNTools that support traditional models of Petri nets.

5 Conclusion

We have presented the Calculus of Covalent Bonding, the Bonding Calculus, and Reversing Petri Nets as models of chemical reactions and reversible processes in general. We have shown that they can all model the out-of-causal-order reversibility present in such reactions. We have also noted that the two process calculi allow few reactions which do not happen in reality. This is due to the modelling that abstracts away from some chemical properties of atoms and molecules such as, for example, spacial arrangement and distance between molecules. In future work, we plan to develop these formalisms further and apply them to the modelling and reasoning about reversible biochemical reactions and processes.

References

1. Aman, B., Ciobanu, G.: Bonding calculus. Natural Computing **17**(4), 823–832 (2018)
2. Baldan, P., Cocco, N., Marin, A., Simeoni, M.: Petri nets for modelling metabolic pathways: a survey. Natural Computing **9**(4), 955–989 (2010)

3. Barylska, K., Gogolińska, A., Mikulski, Ł., Philippou, A., Piątkowski, M., Psara, K.: Reversing computations modelled by coloured Petri nets. In: Proceedings of ATAED 2018. CEUR Workshop Proceedings, vol. 2115, pp. 91–111 (2018)
4. Blätke, M.A., Heiner, M., Marwan, W.: Petri nets in systems biology. Technical Report, Otto-von-Guericke University Magdeburg (2011)
5. Chaouiya, C.: Petri net modelling of biological networks. Briefings in Bioinformatics **8**(4), 210–219 (2007)
6. Ciocchetta, F., Hillston, J.: Bio-PEPA: A framework for the modelling and analysis of biological systems. Theoretical Computer Science **410**(33-34), 3065–3084 (2009)
7. Danos, V., Feret, J., Fontana, W., Harmer, R., Krivine, J.: Rule-based modelling of cellular signalling. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007 - Concurrency Theory, Lecture Notes in Computer Science, vol. 4703, pp. 17–41. Springer (2007)
8. Danos, V., Krivine, J.: Reversible communicating systems. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004 - Concurrency Theory, Lecture Notes in Computer Science, vol. 3170, pp. 292–307. Springer (2004)
9. Faeder, J.R., Blinov, M.L., Hlavacek, W.S.: Rule-based modeling of biochemical systems with BioNetGen. Methods in Molecular Biology **500**, 113–167 (2009)
10. Fages, F., Soliman, S., Chabrier-Rivier, N.: Modelling and querying interaction networks in the biochemical abstract machine BIOCHAM. Journal of Biological Physics and Chemistry **4**, 64–73 (2004)
11. Fokkink, W.: Introduction to Process Algebra. Springer (2000)
12. Hofestädt, R.: A Petri net application of metabolic processes. Journal of System Analysis, Modeling and Simulation **16**, 113–122 (1994)
13. Hofestädt, R., Thelen, S.: Quantitative modeling of biochemical networks. In Silico Biology **1**(1), 39–53 (1998)
14. Kuhn, S.: Simulation of base excision repair in the calculus of covalent bonding. In: Karri, J., Ulidowski, I. (eds.) Reversible Computation, 10th International Conference, RC 2018, Lecture Notes in Computer Science, vol. 11106, pp. 123–129. Springer (2018)
15. Kuhn, S., Ulidowski, I.: A calculus for local reversibility. In: Devitt, S., Lanese, I. (eds.) Reversible Computation, 8th International Conference, RC 2016, Lecture Notes in Computer Science, vol. 9720, pp. 20–35. Springer (2016)
16. Kuhn, S., Ulidowski, I.: Local reversibility in a calculus of covalent bonding. Science of Computer Programming **151**(Supplement C), 18–47 (2018)
17. Lanese, I., Mezzina, C.A., Schmitt, A., Stefani, J.B.: Controlling reversibility in higher-order pi. In: Katoen, J.P., König, B. (eds.) CONCUR 2011 - Concurrency Theory, Lecture Notes in Computer Science, vol. 6901, pp. 297–311. Springer (2011)
18. Lanese, I., Mezzina, C.A., Stefani, J.B.: Controlled reversibility and compensations. In: Glück, R., Yokoyama, T. (eds.) Reversible Computation, 4th International Workshop, RC 2012, Lecture Notes in Computer Science, vol. 7581, pp. 233–240. Springer (2013)
19. Lanese, I., Mezzina, C.A., Stefani, J.B.: Reversing higher-order pi. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010 - Concurrency Theory. Lecture Notes in Computer Science, vol. 6269, pp. 478–493. Springer (2010)
20. Matsuno, H., Nagasaki, M., Miyano, S.: Hybrid Petri net based modeling for biological pathway simulation. Natural Computing **10**(3), 1099–1120 (2011)
21. Milner, R.: A Calculus of Communicating Systems, Lecture Notes in Computer Science, vol. 92. Springer (1980)

22. Peleg, M., Rubin, D.L., Altman, R.B.: Using Petri net tools to study properties and dynamics of biological systems. Journal of the American Medical Informatics Association **12**(2), 181–199 (2005)
23. Philippou, A., Psara, K.: Reversible computation in petri nets. In: Kari, J., Ulidowski, I. (eds.) Reversible Computation - 10th International Conference, RC 2018, Leicester, UK, September 12-14, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11106, pp. 84–101. Springer (2018)
24. Philippou, A., Psara, K., Siljak, H.: Controlling reversibility in reversing Petri nets with application to wireless communications. In: Thomsen, M.K., Soeken, M. (eds.) Reversible Computation - 11th International Conference, RC 2019. Lecture Notes in Computer Science, vol. 11497, pp. 238–245. Springer (2019)
25. Phillips, I., Ulidowski, I.: Reversing algebraic process calculi. The Journal of Logic and Algebraic Programming **73**(1-2), 70–96 (2007)
26. Phillips, I., Ulidowski, I.: Reversibility and asymmetric conflict in event structures. In: D'Argenio, P.R., Melgratti, H. (eds.) CONCUR 2013 - Concurrency Theory, Lecture Notes in Computer Science, vol. 8052, pp. 303–318. Springer (2013)
27. Phillips, I., Ulidowski, I., Yuen, S.: Modelling of bonding with processes and events. In: Dueck, G.W., Miller, D.M. (eds.) Reversible Computation, 5th International Conference, RC 2013, Lecture Notes in Computer Science, vol. 7948, pp. 141–154. Springer (2013)
28. Phillips, I., Ulidowski, I., Yuen, S.: A reversible process calculus and the modelling of the ERK signalling pathway. In: Glück, R., Yokoyama, T. (eds.) Reversible Computation, 4th International Workshop, RC 2012, Lecture Notes in Computer Science, vol. 7581, pp. 218–232. Springer (2013)
29. Popova-Zeugmann, L., Heiner, M., Koch, I.: Time Petri nets for modelling and analysis of biochemical networks. Fundamenta Informaticae **67**(1-3), 149–162 (2005)
30. Priami, C.: Stochastic π-calculus. The Computer Journal **38**(7), 578–589 (1995)
31. Priami, C., Quaglia, P.: Beta binders for biological interactions. In: Danos, V., Schachter, V. (eds.) Computational Methods in Systems Biology. pp. 20–33. Springer (2005)
32. Reddy, V.N., Mavrovouniotis, M.L., Liebman, M.N.: Petri net representations in metabolic pathways. In: Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology. pp. 328–336. AAAI (1993)
33. Regev, A., Panina, E.M., Silverman, W., Cardelli, L., Shapiro, E.: Bioambients: an abstraction for biological compartments. Theoretical Computer Science **325**(1), 141–167 (2004)
34. Regev, A., Shapiro, E.: The π-calculus as an abstraction for biomolecular systems. In: Ciobanu, G., Rozenberg, G. (eds.) Modelling in Molecular Biology. pp. 219–266. Springer (2004)
35. Reisig, W.: Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies. Springer (2013)
36. Ulidowski, I.: Equivalences on observable processes. In: Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science. pp. 148–159. IEEE (1992)
37. Ulidowski, I., Phillips, I., Yuen, S.: Reversing event structures. New Generation Comput. **36**(3), 281–306 (2018)
38. Voss, K., Heiner, M., Koch, I.: Steady state analysis of metabolic pathways using Petri nets. In Silico Biology **3**(3), 367–387 (2003)